Verification versus Validation

In real-time embedded automotive software

François-Xavier de LAUNET

April 2015

Agenda

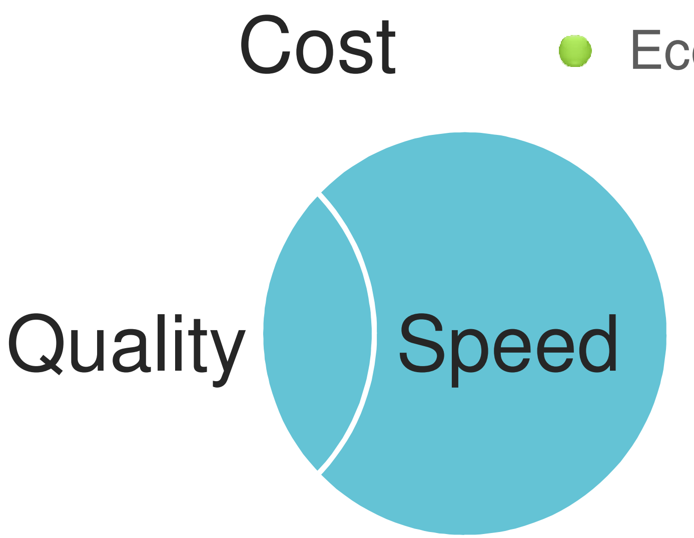
1 Tests usual/specific issues in real-time embedded automotive software - Context

2 Optimize test effort during design process - Triptych

3 Test and verification techniques portfolio - Examples

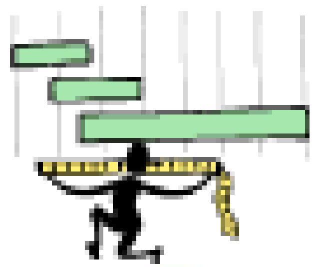
francois-xavier.de-launet@valeo.com

Software automotive constraints

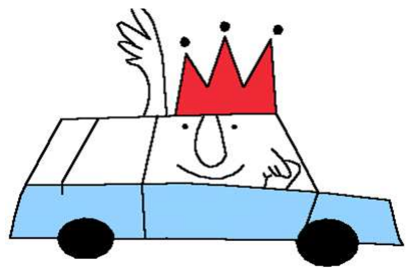


- Stay in cost frame (R&D budget)
- Project profitability (Margin)
- Eco design (Limit HW resources usage, commercially viable)

- Automotive industry is planning-driven *(Start Of Production never shift)*
- Strong VALEO commitment but Customer needs uncertainty



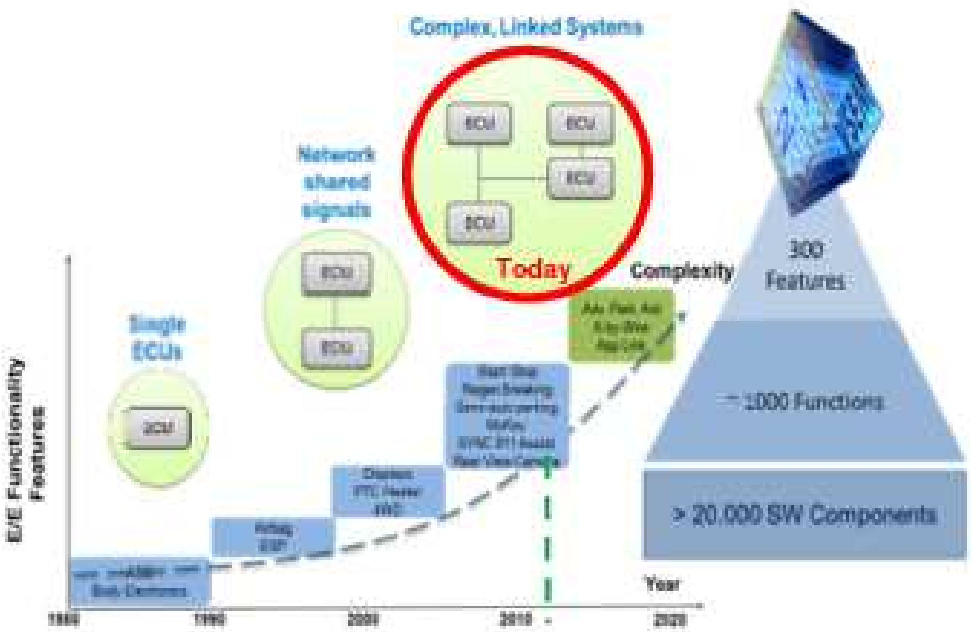
- *Economic Security* (Mass production)
- Complexity increase due to innovations in Premium Cars
- Dependability *(ISO 26262)*
- Deliver demanded functionality without trial & errors



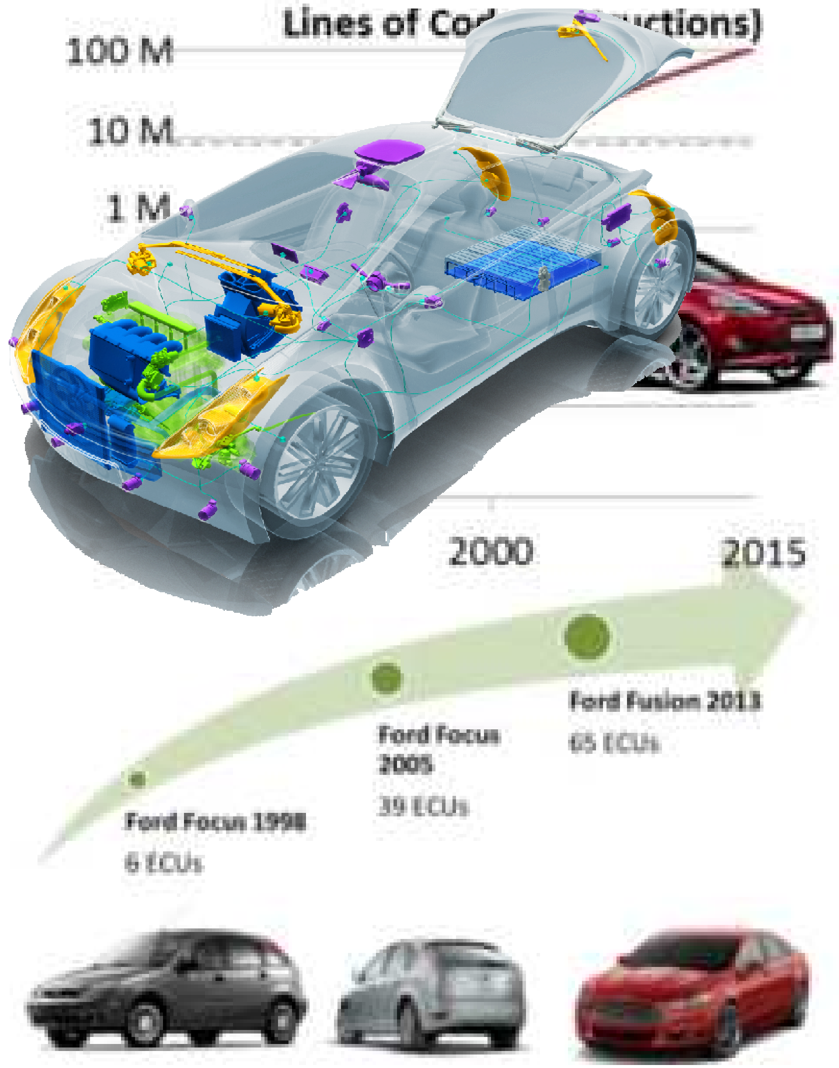
Valeo

Software automotive increased complexity

Features are highly complex and often distributedover many physical modules



Growth of Software and increasing complexity require enhanced Processes, Methods&Tools to support development of high quality software.



Source : FORD QPIP11C

VALEO at a glance

- VALEO: Focus innovation on reducing CO_2 emissions and is a pioneer in “Intuitive Driving”.

At the end of the day…

… car will be automated,

AUTOMATED CARS

SAFE & INTUITIVE HMI

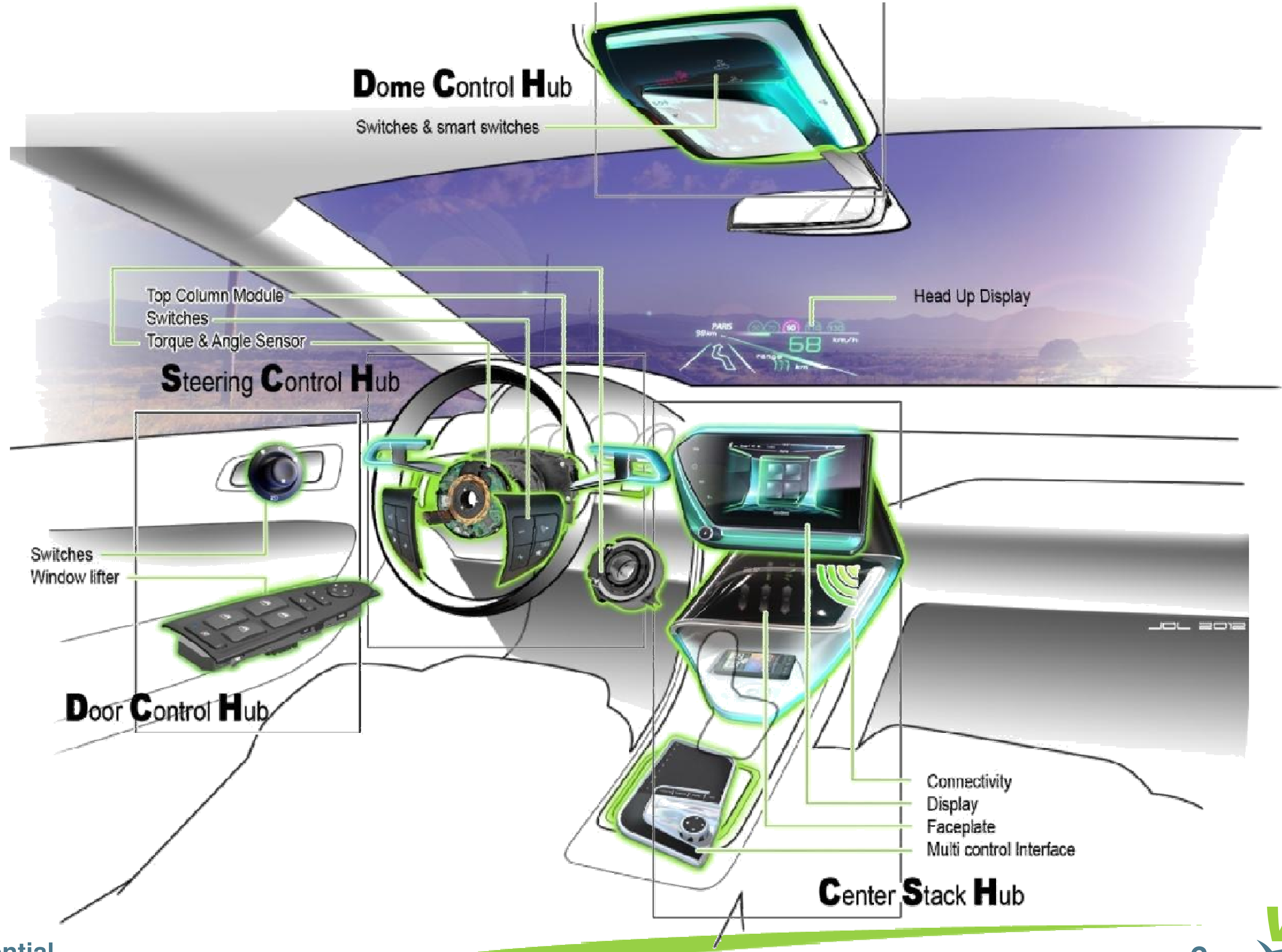
…car will be connected,

…and new human machine interfaces will be required

VALEO INTUITIVE DRIVING

CONNECTED CARS

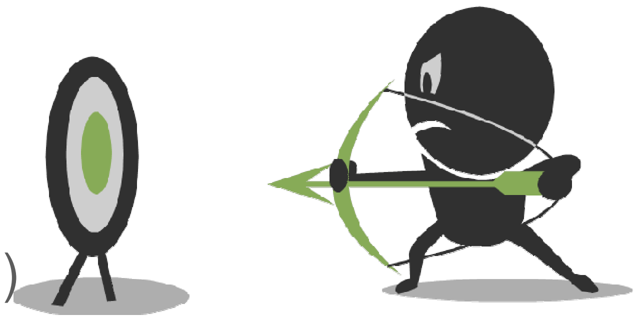
VALEO Intuitive Driving Cockpits



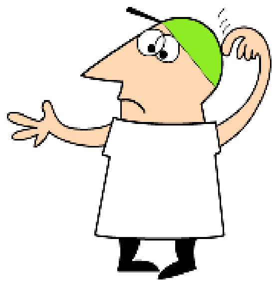
Test usual issue - What keeps us up at night ?

- Project Planning
  - Previous design phases may overrun
  - Time-to-market pressures affect time available for testing
  - Limited resources
- Complete testing NOT feasible
  - Increased complexity
  - Real-time non-determinism

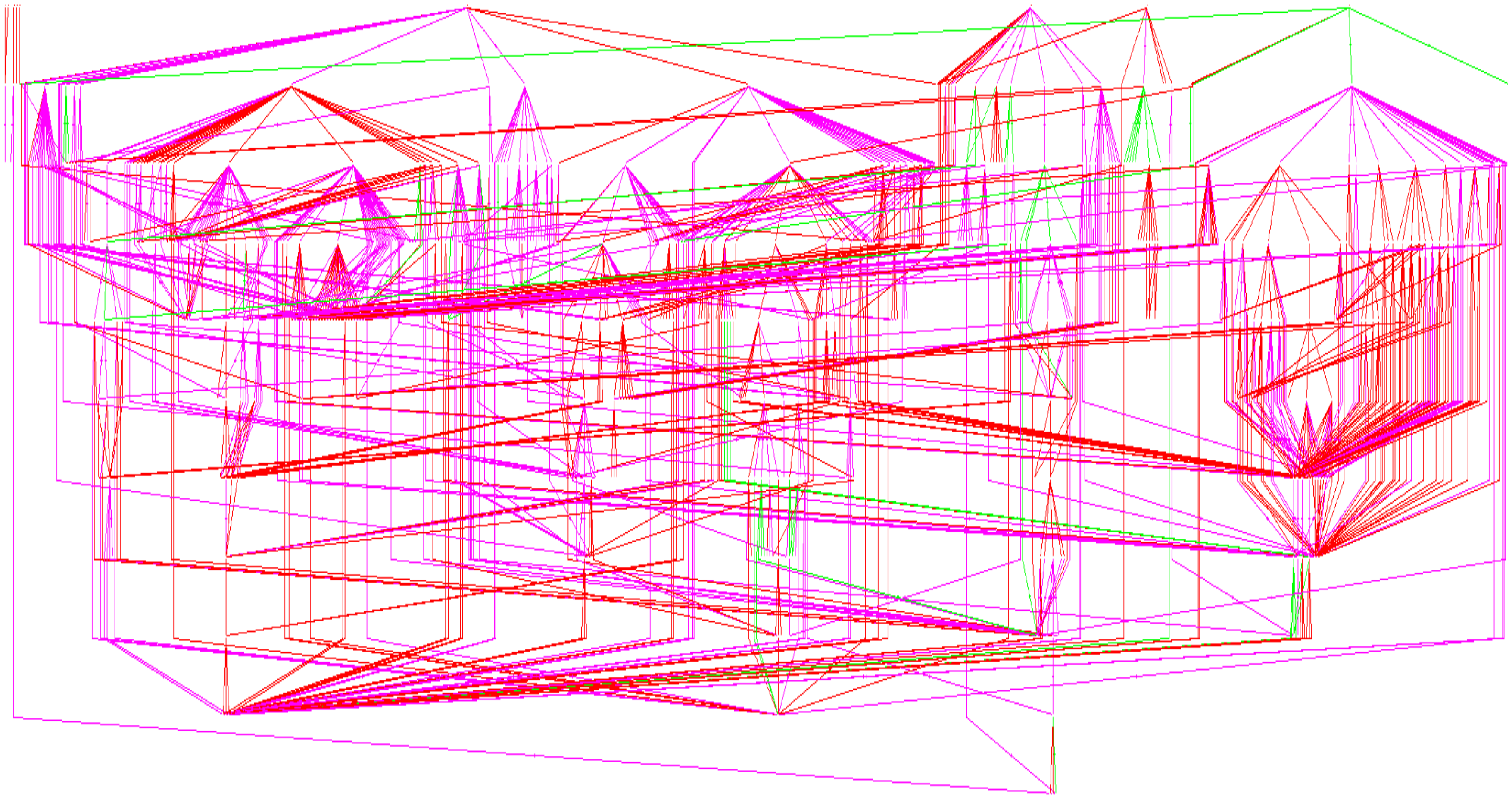
- Appropriate Testing
  - Economic security
  - Norm requirements (Dependability)
  - Effective testing; Non redundant testing
  - Quantifiable (How much test and where?)
  - Realistic and commercially viable



Test usual issues – Quality control



- How to guarantee the quality of this software?



Test usual issues - Conclusion

- Software verification & Validation is satisfied thru:
  - development of test cases and procedures, and subsequent execution of those test procedures
  - a combination of reviews, inspections, analysis, walkthroughs, check-list, part of software engineering activities

It is not simply testing.

Testing, in general,
cannot show the absence of errors.

Agenda

2 Optimize test effort during design process - Triptych

Cut the complexity

Architecture driven

Foster reuse (Idea of "*Fixed – Variable – New*")

3 Test and verification techniques portfolio - Examples

4 Conclusion

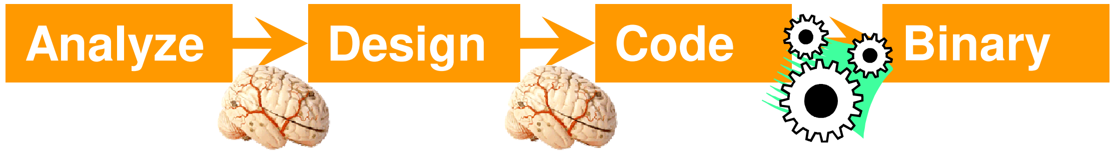
francois-xavier.de-launet@valeo.com

Why do we need Software Processes?

- Human cognitive complexity
  - Miller law (1956)

The capacity of our short term memory is 7 (+ or – 2) « Chunks »

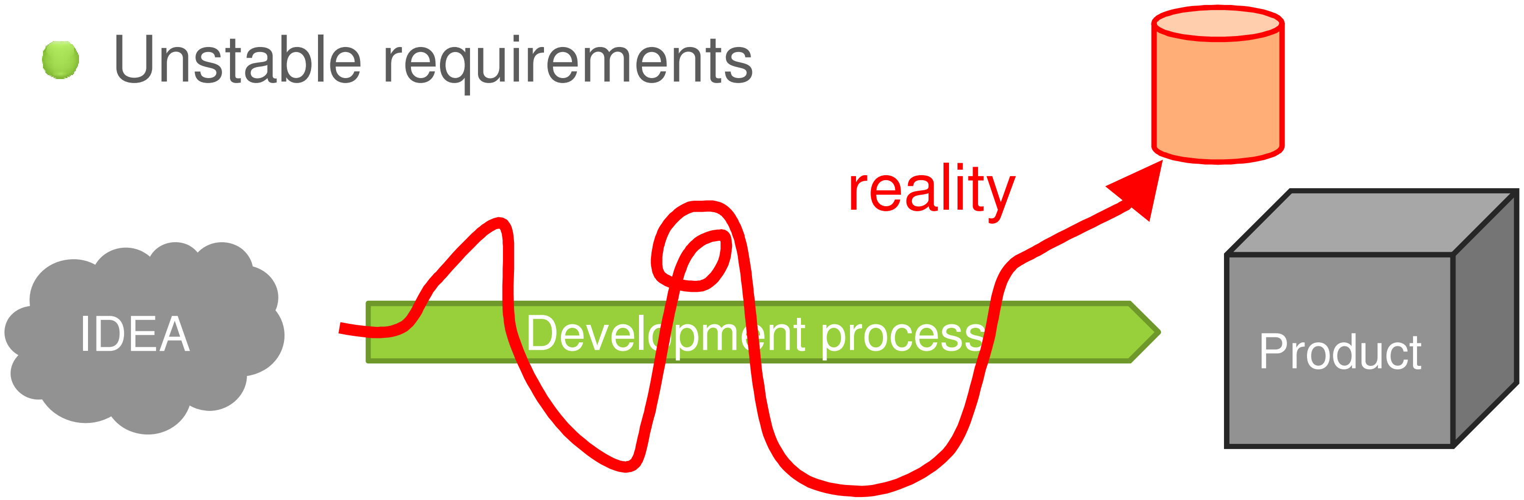
  - Software Design



➔ Human comprehension => **Cognitive complexity**

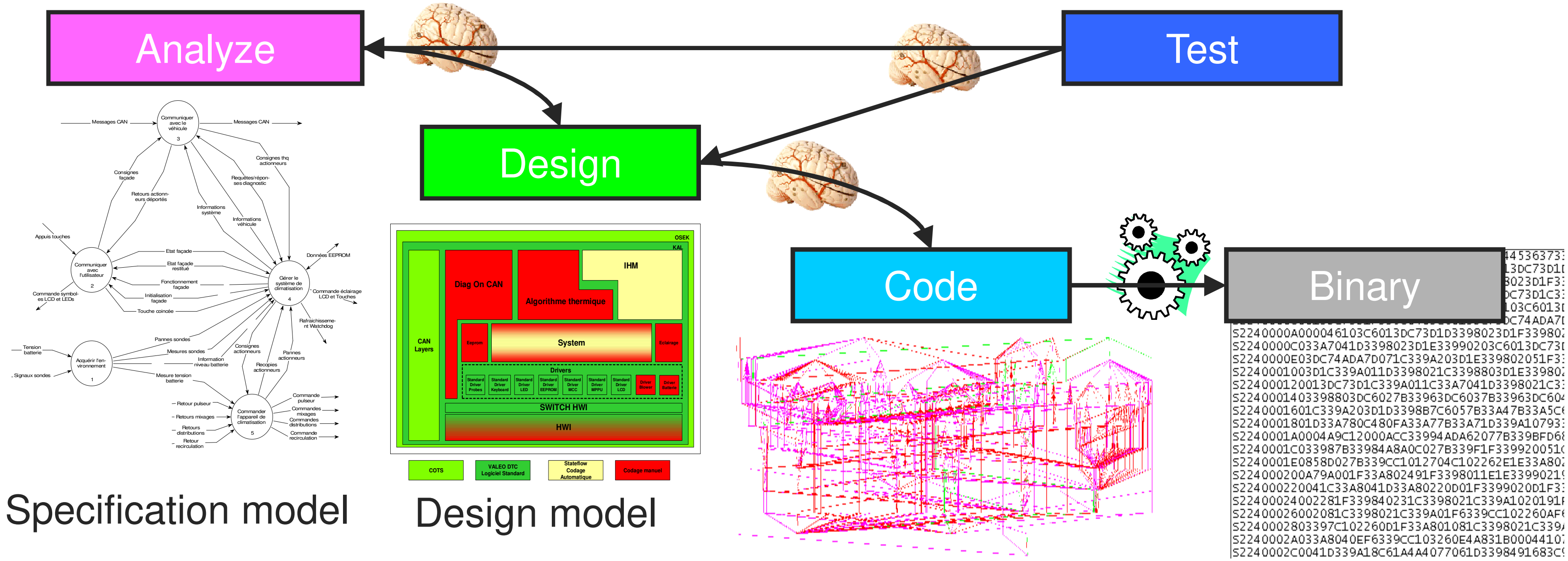
➔ Real time execution, data exchange => **Relation complexity**

- Unstable requirements



Software Quality Concept

- Software Complexity harnessing is needed
  - Because SW Design is Human; Human cognitive capacity is limited
  - Reducing complexity at each step « guarantee » SW quality
  - Thanks to appropriate organization, process, technical design rules



Specification model

Design model

Software Quality is achieved by reducing complexity at each PROCESS STEP

Vehicle development process

OEM determines vehicle

Vehicle functions
Vehicle SOP & price

OEM defines parts

Part specification / requirements
Parts supplier / costs

OEM defines milestones

Sample definition (scope & quality)

OEM validates vehicle

Part acceptance

OEM integrate parts

Issue: inconsistencies between specifications and requirements

OEM collect sample

Check for inconsistencies
Partial integration

OEM defines change

Request solution
Decide on scope of change

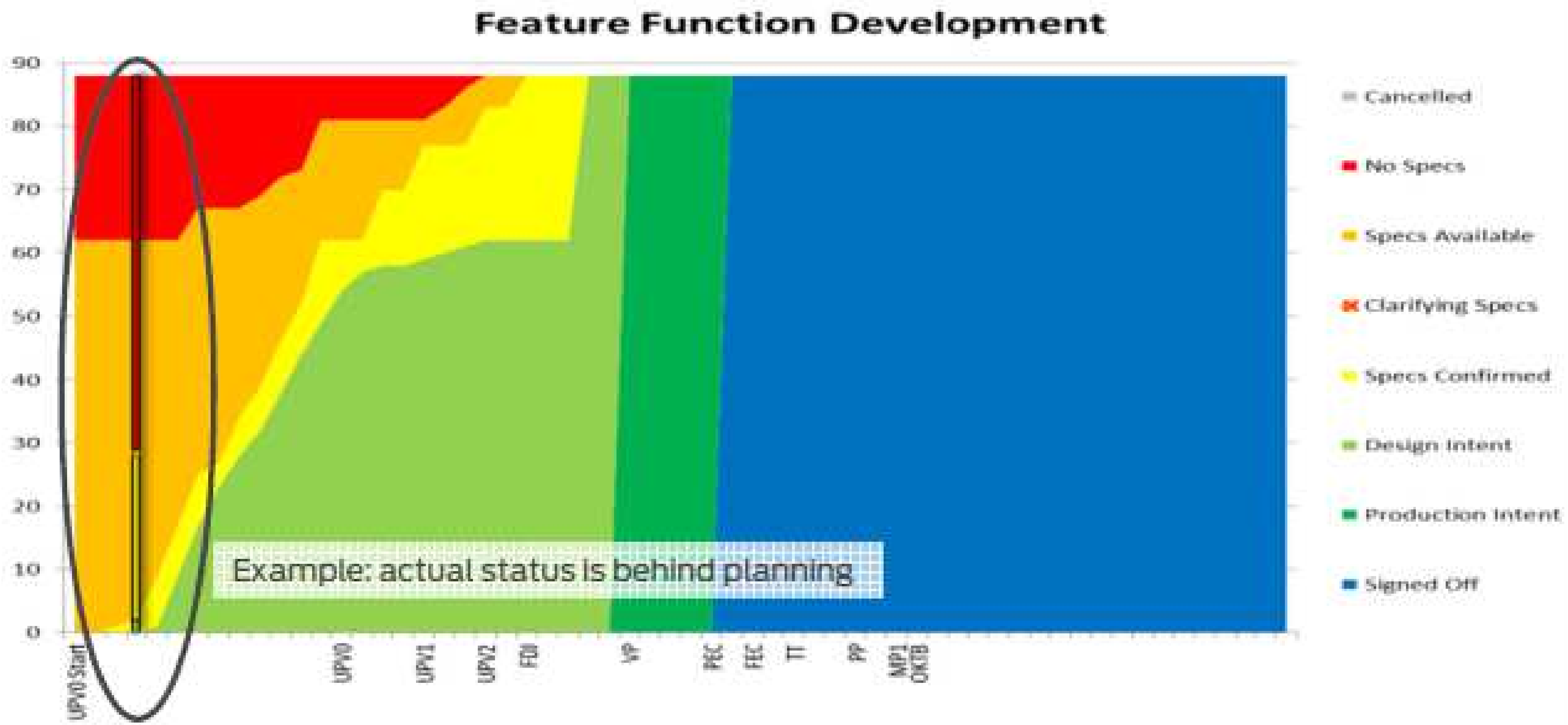
Supplier develop part

Requirements Analysis, determine sub parts, determine reuse, develop new sub parts, integrate sub parts, validate parts
Develop sample, support milestones

Process is Iterative in Nature and OEMs need to start somewhere

Valeo

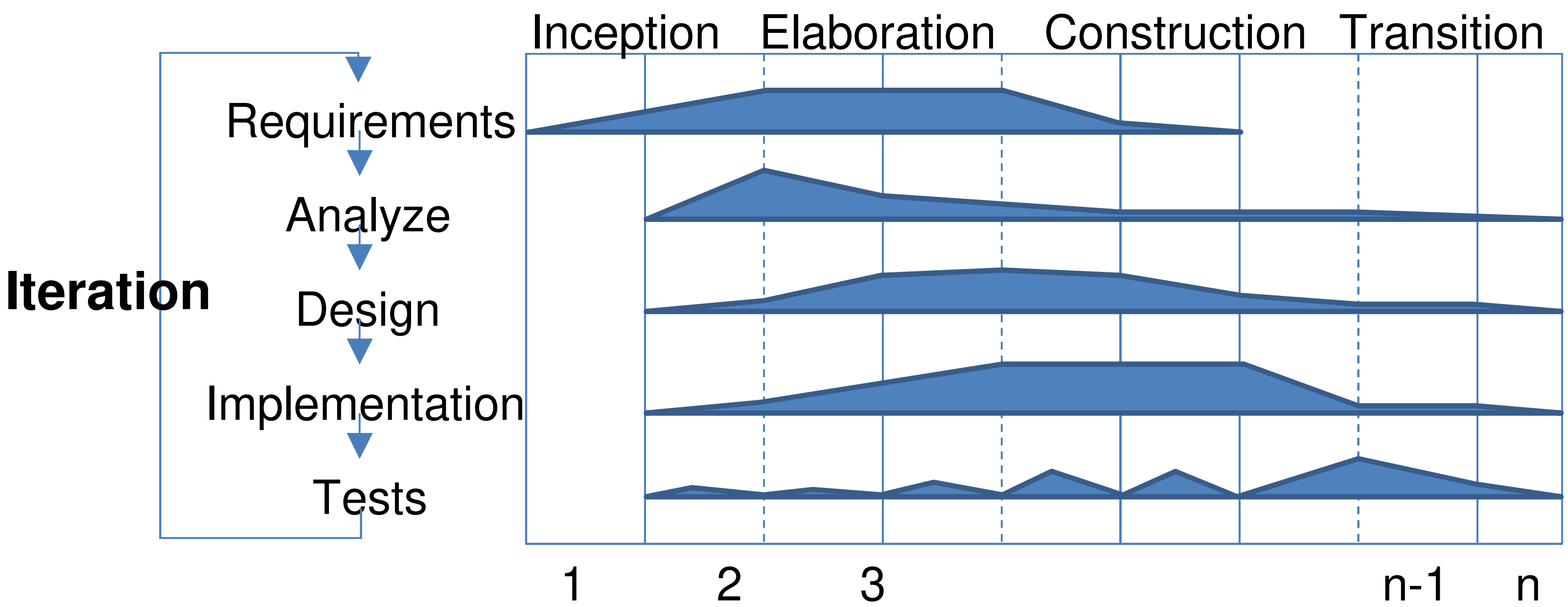
Vehicle development process - Example



Source : FORD QPIP11C - Supplier_Reports_Supplier_presentation

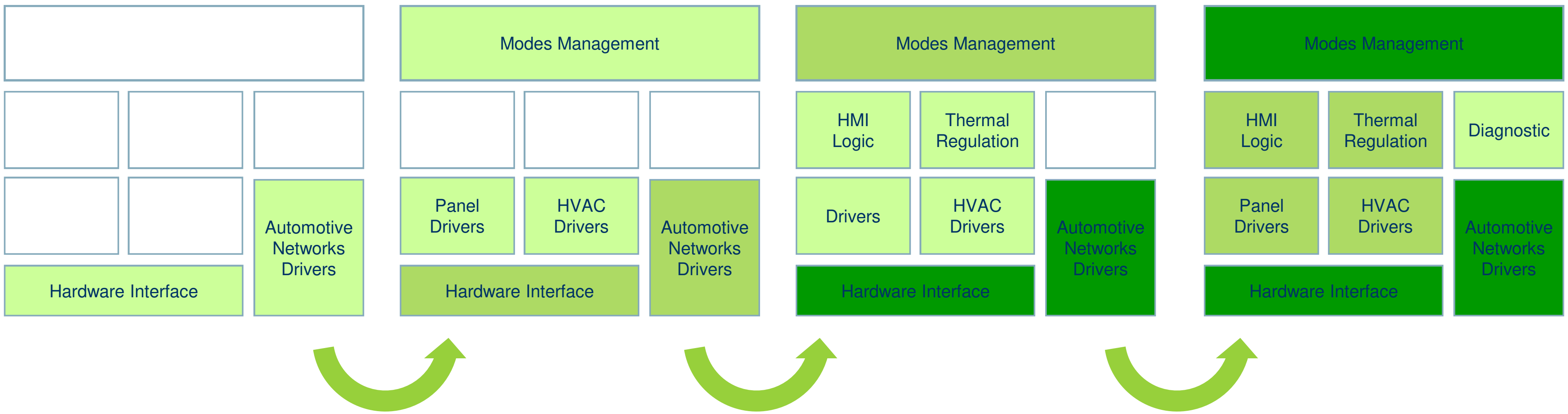
Iterative/incremental approach

- Cut the process complexity using iterative approach
  - Manage small work package
  - Deliver frequently to have feedback
  - Bring closer all V-Cycle activities
  - Limit work in progress
  - Match to customer feature implementation plan

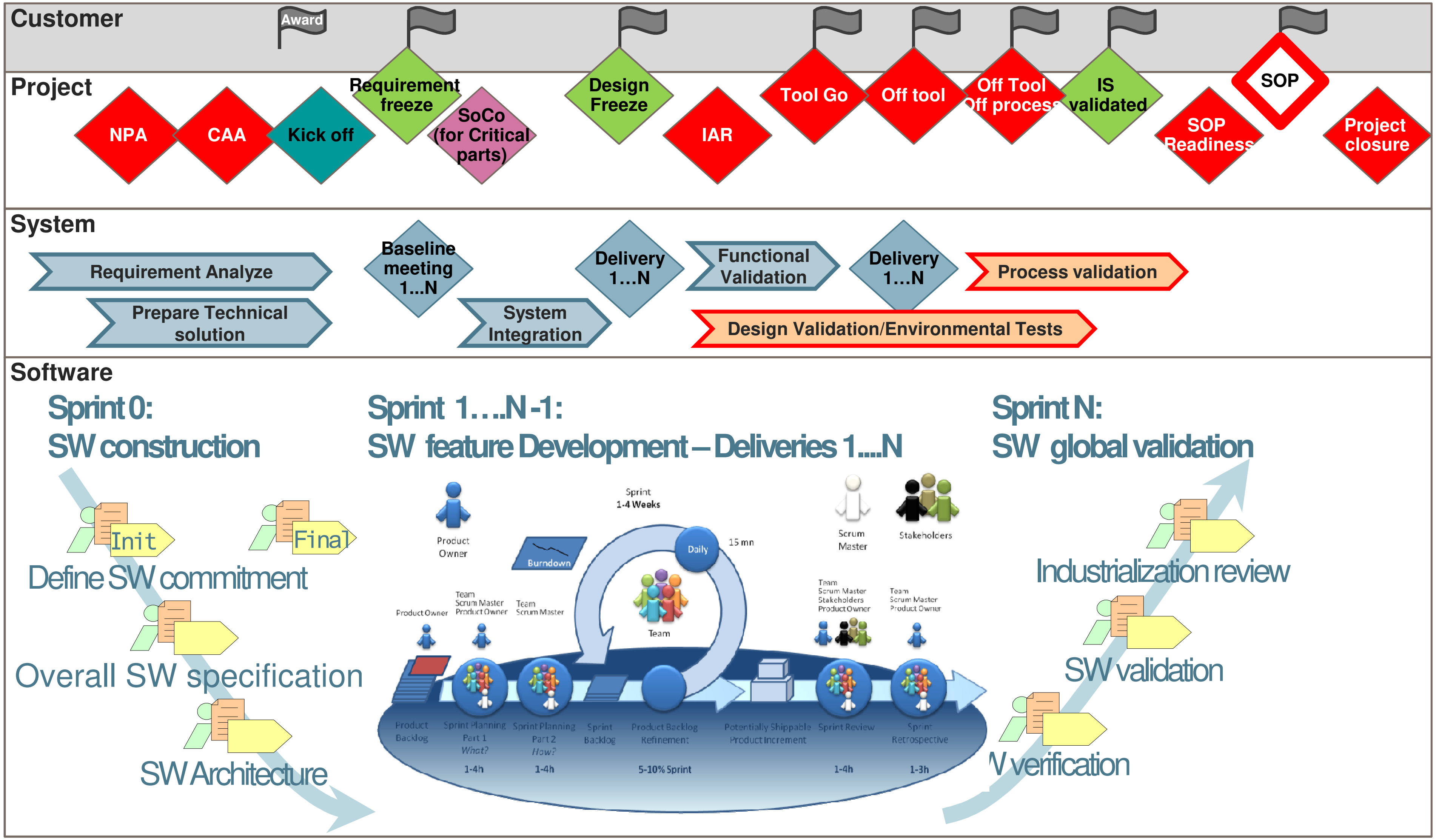


Iterative/incremental approach

- Cut the product complexity using incremental approach
  - Validate first technical and architectural assumptions
  - First indentify hard real time and material constraints
  - Do more often what hurts the most (continuous integration, test immediately)
  - Monitor technical debt
  - Demonstrate product quality before to go to mass production



VALEO Incremental/Iterative process



Basics for test optimization

- Strategy for improvement

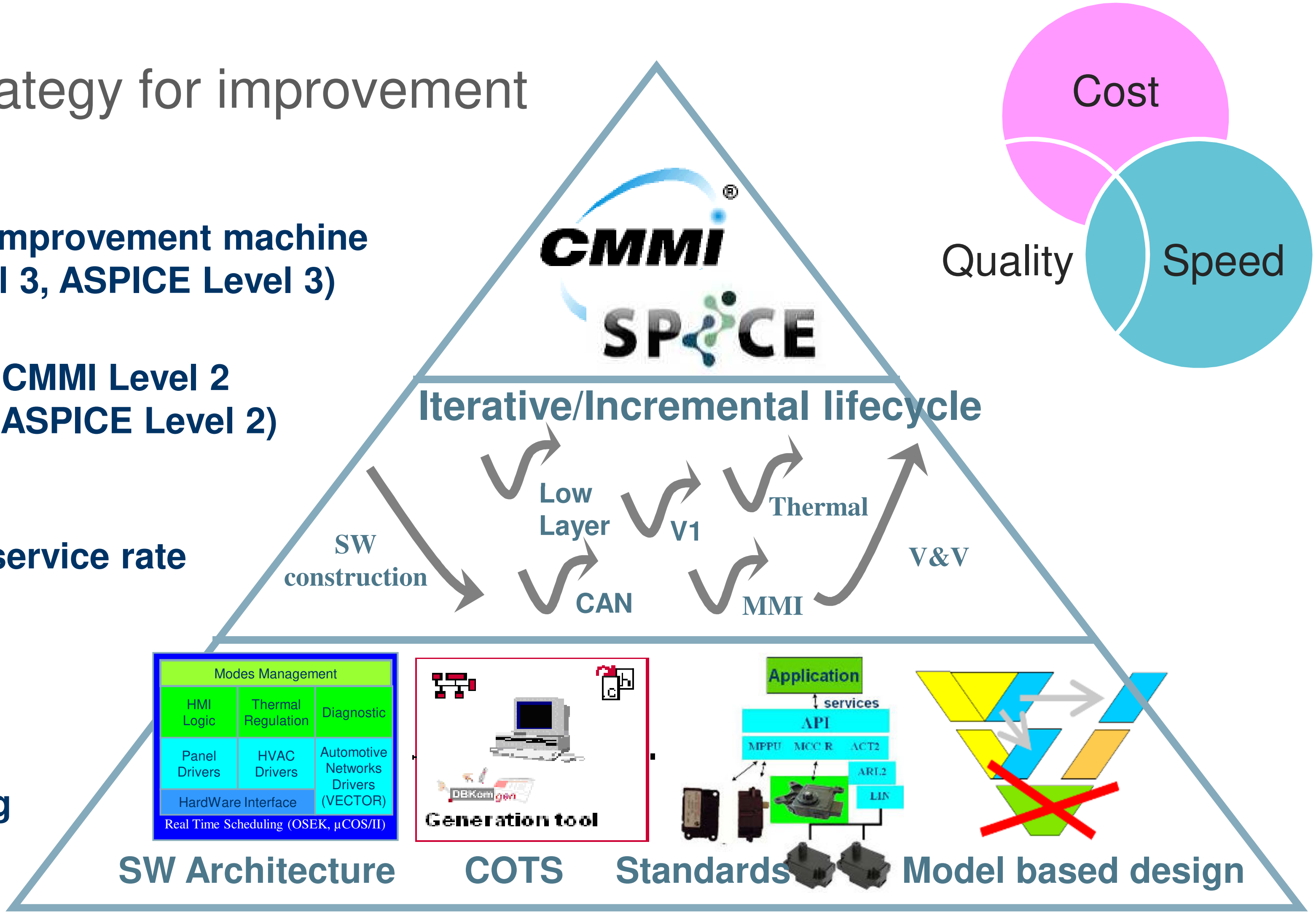
Efficiency, improvement machine (CMMI Level 3, ASPICE Level 3)

Discipline (CMMI Level 2 ASPICE Level 2)

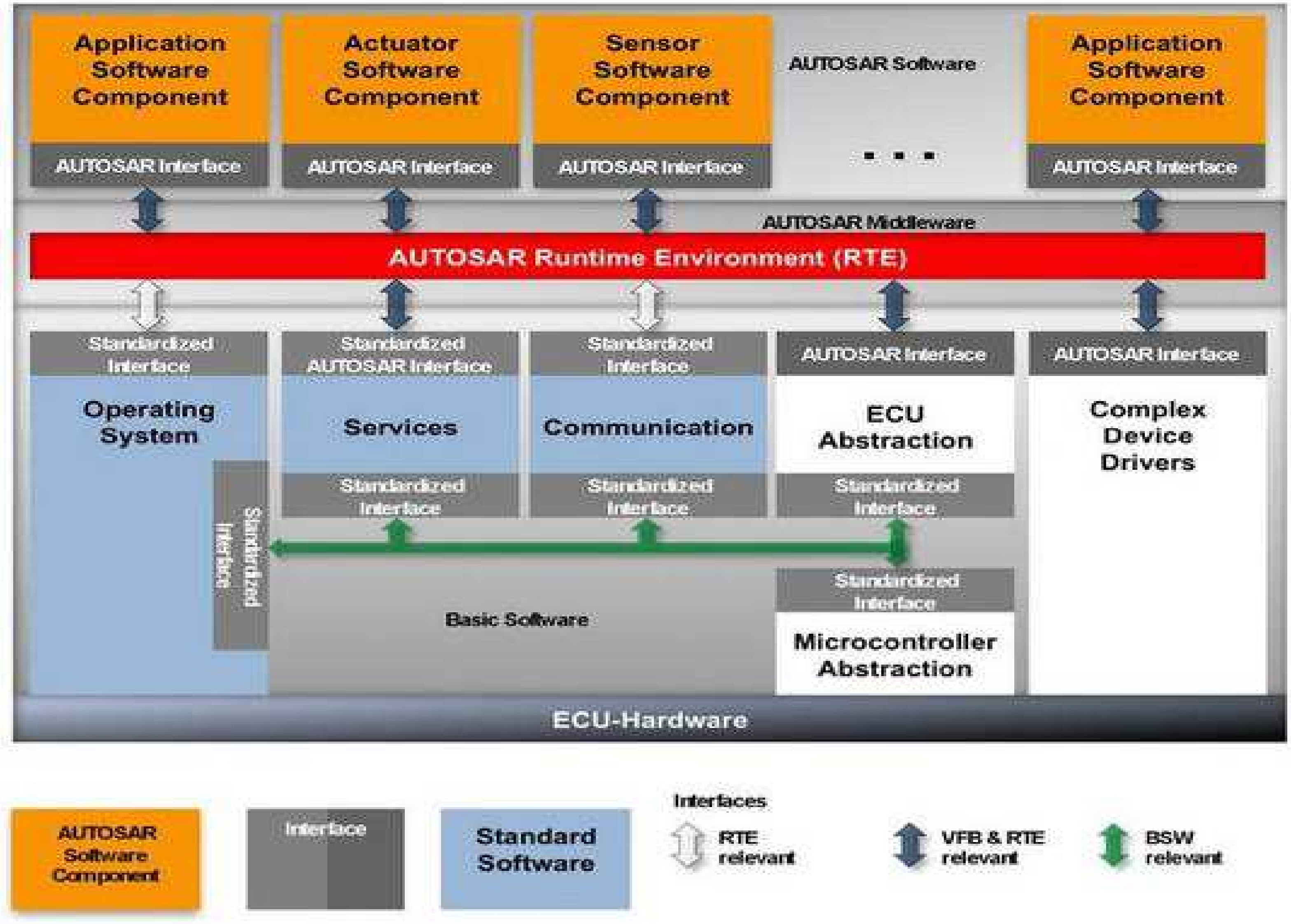
Customer, service rate

Software Engineering

Start



Standard Architecture



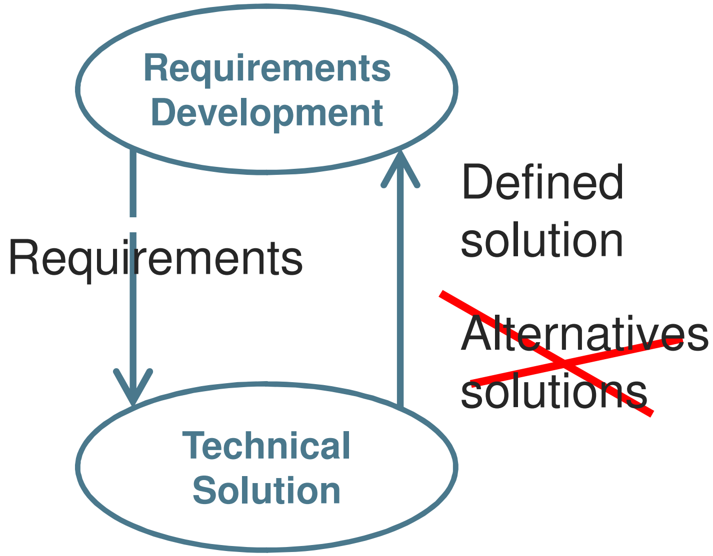
Requirement development approach

- Cut the complexity, thinking to architecture
  - CMMI – Requirements Development (RD) Practices

SG 1 Develop Customer Requirements
- SP 1.1 Elicit Needs
- SP 1.2 Transform Stakeholder Needs into Customer Requirements

SG 2 Develop Product Requirements
- SP 2.1 Establish Product and Product Component Requirements
- SP 2.2 Allocate Product Component Requirements
- SP 2.3 Identify Interface Requirements

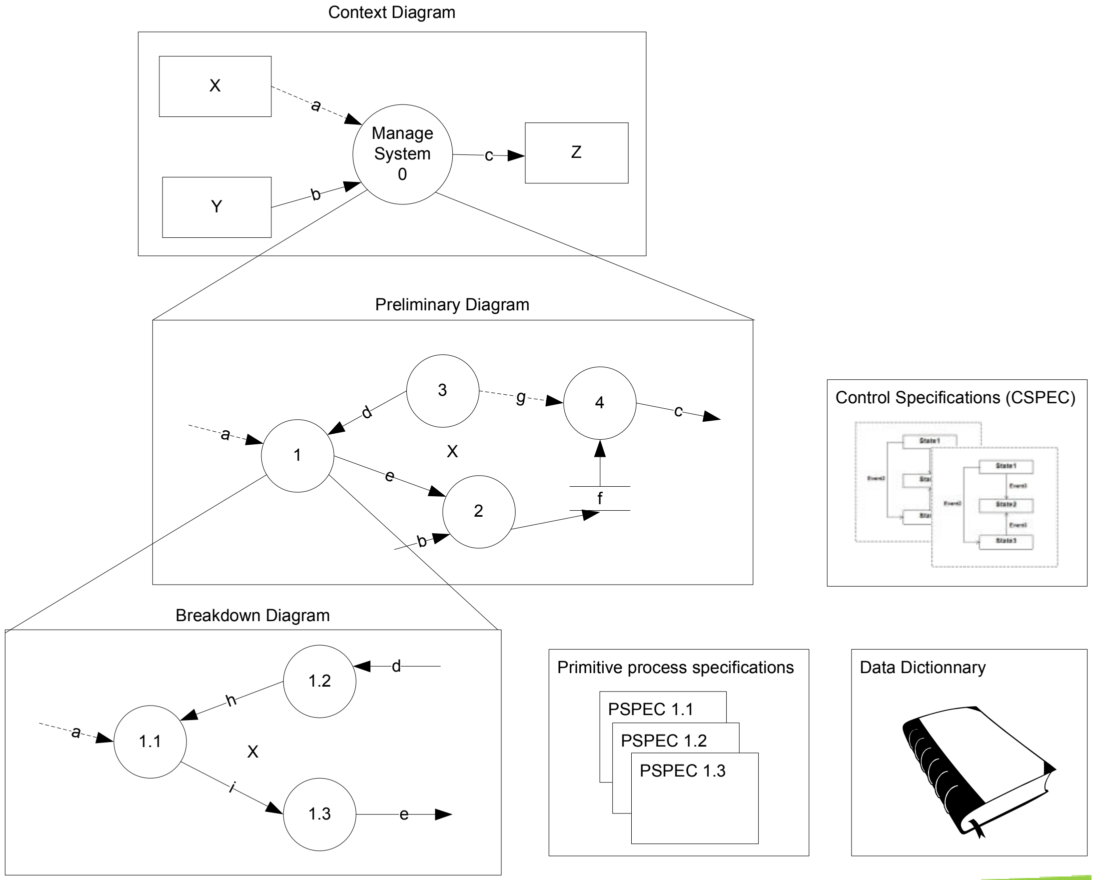
SG 3 Analyze and Validate Requirements
- SP 3.1 Establish Operational Concepts and Scenarios
- SP 3.2 Establish a Definition of Required Functionality and Quality Attributes
- SP 3.3 Analyze Requirements
- SP 3.4 Analyze Requirements to Achieve Balance
- SP 3.5 Validate Requirements



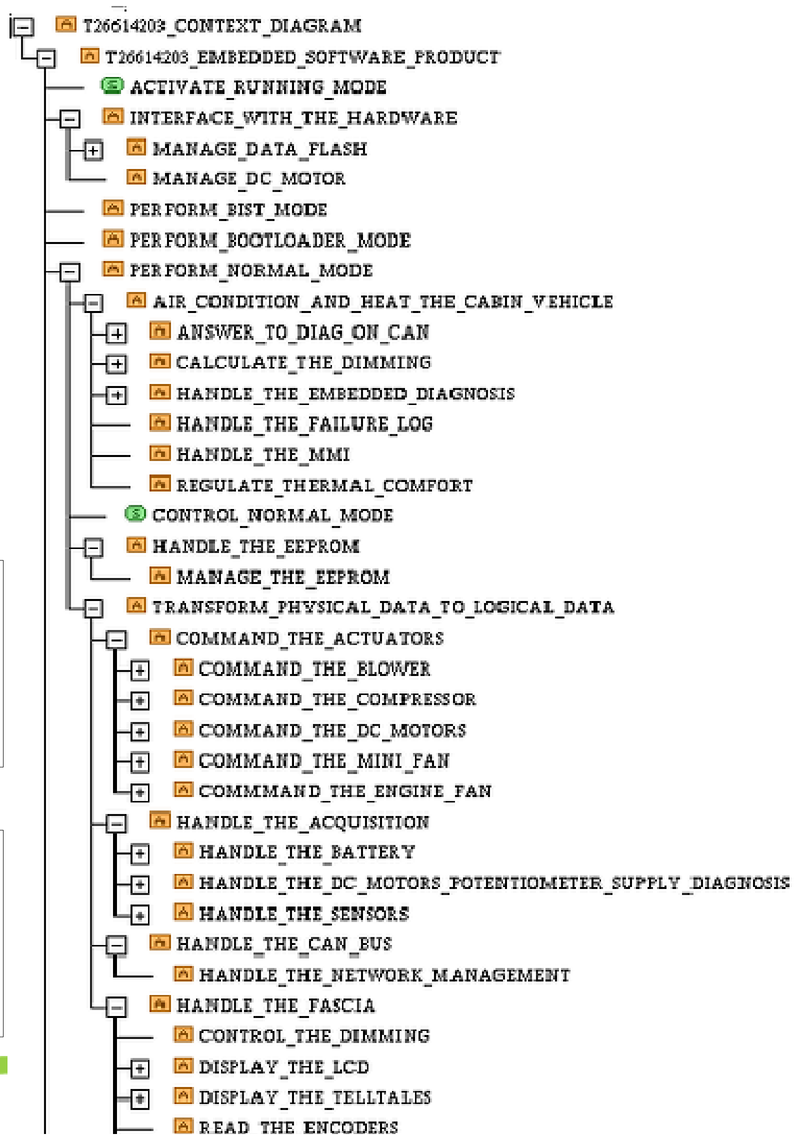
- Architecture Driven
  - Early technical solutions…..
    …..flexible to support iteration
  - Influenced by standard architecture
    experience of the team

Requirement analysis (1/2)

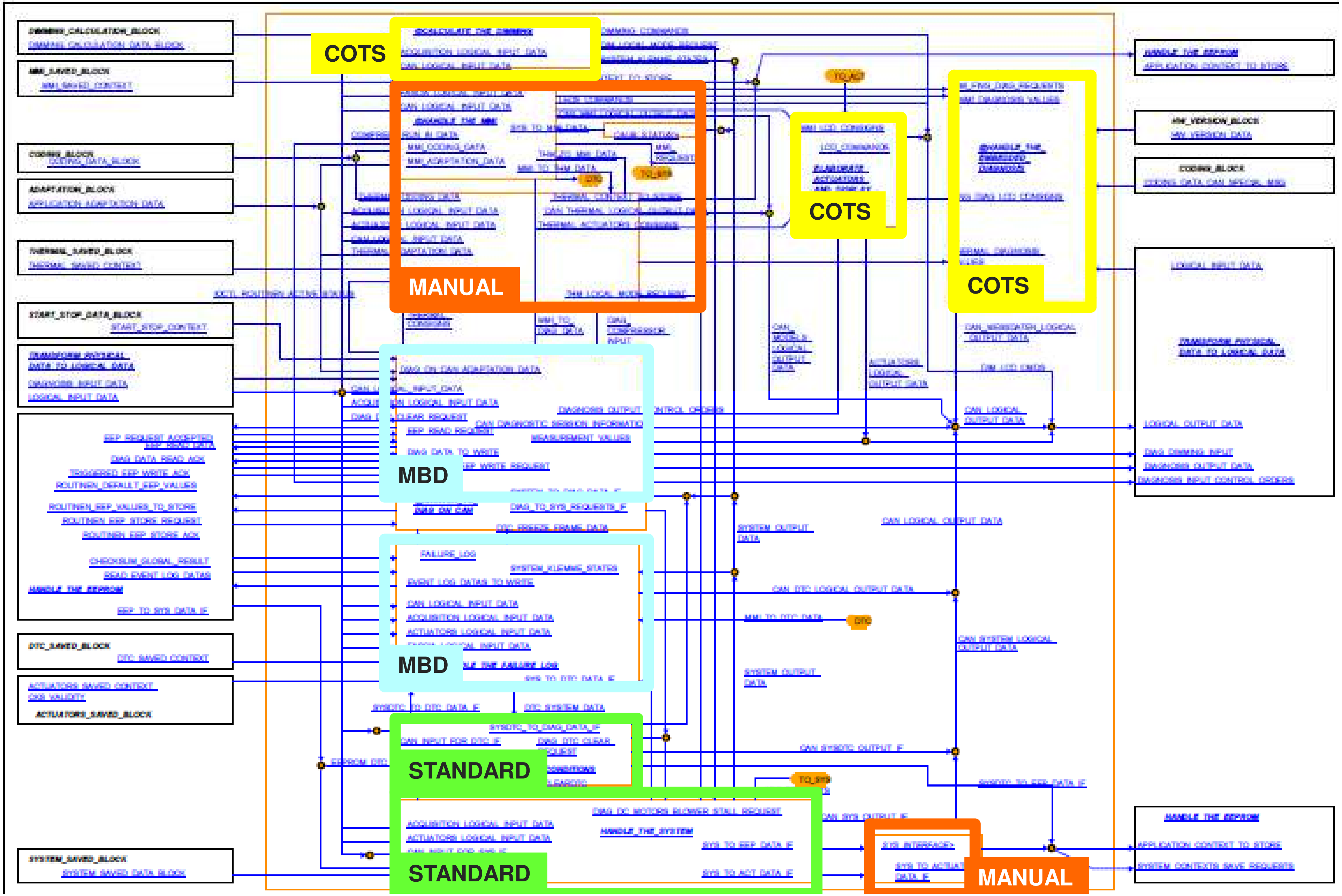
- Fonctional approach
  - From Context diagram…
  - … to full SART model



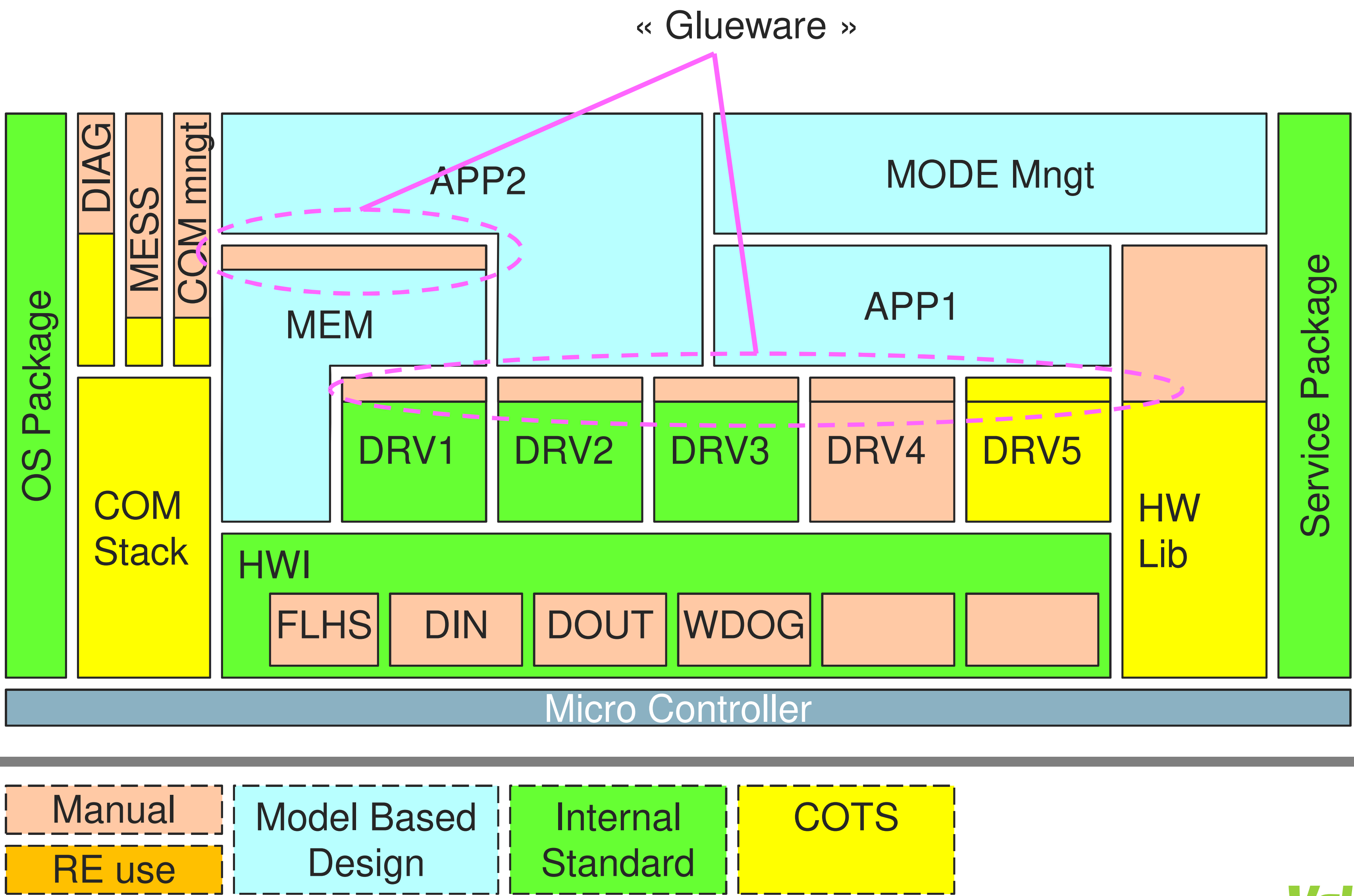
Architecture oriented



Requirement analysis (2/2)



Standard Architecture – Component type (1/2)



Standard Architecture – Component type (2/2)

- Component typology:
  - Manual coding (Component developed by the team for the project)
  - Internal standards components (Standards)
  - Component of the Shelf (COTS)
  - Model Based Design (MDB)

  + « Glue ware » : Coding of interface between components

- Purpose is to have simple « Glue ware »
  - No functional requirement allocated (No intelligence inside)
  - Implement process interfaces only (Mapping, scaling, multiplex, event management…)
  - Implement standard/COTS configuration

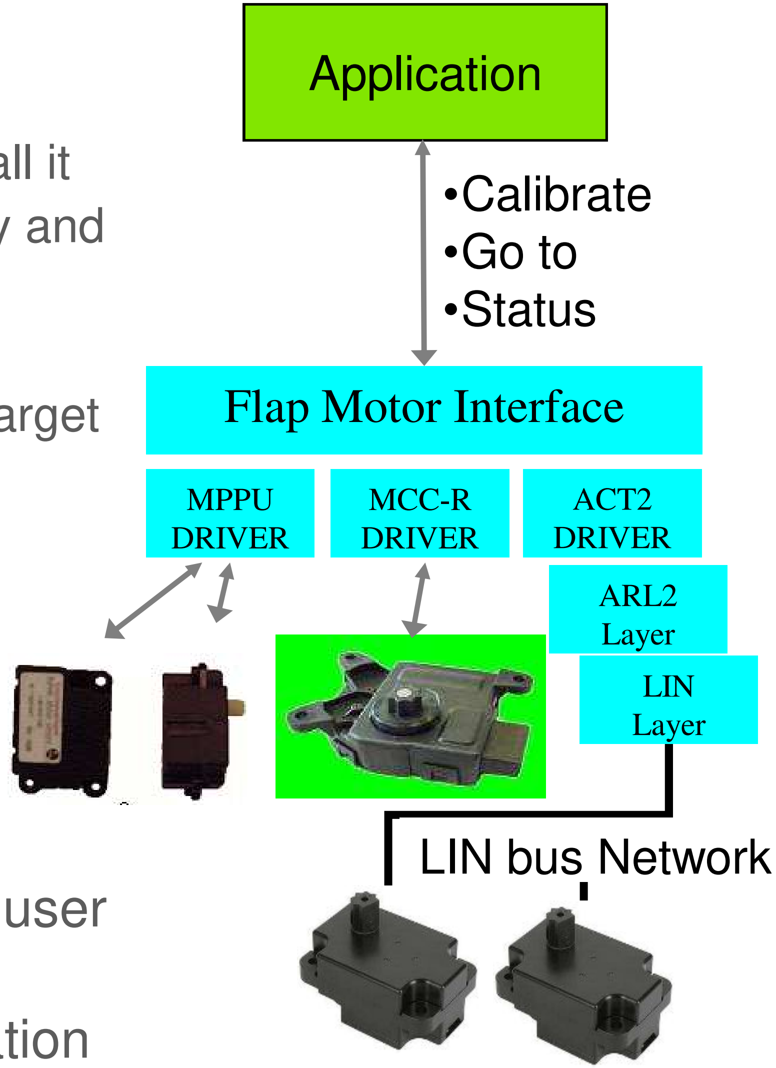
Manual coding – Tests strategy

- Test Objectives
  - Component Unit Tests
    - 100% Modified Condition Decisions coverage
    - « Glue ware », Diagnosis (Garage) functions excluded.
  - Component integration tests
    - Design constraints; Integration constraints; Configuration tested
    - All interfaces tested (API, shared resources, component used)
    - Component performance (Efficiency, Initialization, dynamic behavior)
  - Functional Validation
    - 100% requirements tested using equivalence class method
    - Regression tests before delivery to functional validation
      =>1 nominal test per implemented feature and external interfaces

- Test practices
  - Unit test part of coding activity (Automatic Test campaign)
  - Requirements are categorized and analyzed for correctness and testability

Standard components – Impact on tests

- Standard component : 3 parts
  - API : standard interface :
    - Defines the function and the way to call it
    - Stable in time: it masks the complexity and the evolutions of the implementation
  - Implementation (Core) :
    - Realizes the function for a particular target
    - Incorruptible (Special archiving rules)
  - Variable part :
    - Tunes the standard to project needs



- Impact on Tests
  - Unit Tests to run on µC target only
  - Integration Test plan delivered with user manual
  - Functional tests limited to configuration

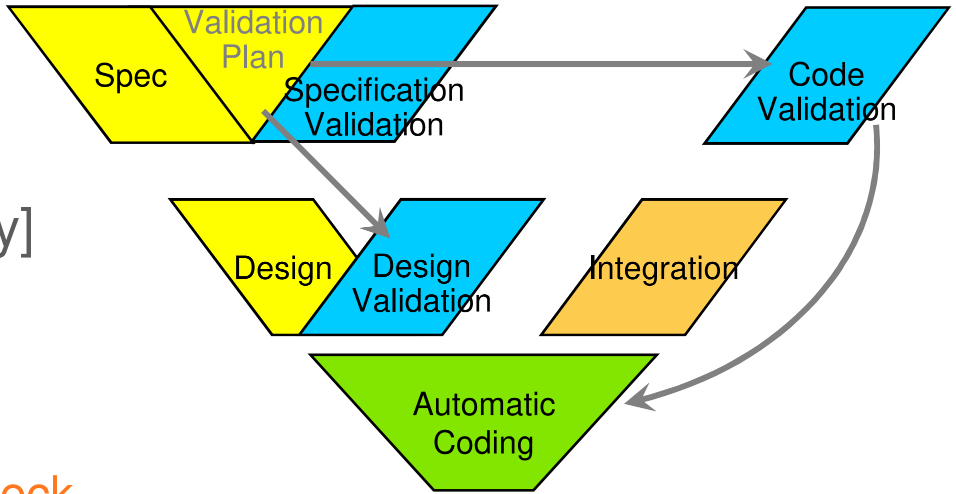
COTS – Impact on tests

- Usage of Components on the shelf (COTS)
  - For OS (OSEK norm)
  - For Communication stacks (CAN, LIN, FLEXRAY norms)
  - For Autosar components (Memory stack…)

- Impact on test
  - Unit Tests are done by Supplier of COTS
    - Par of supplier delivery acceptance (SAM process)
  - Integration tests checks all configuration
  - Functional tests based on Test Suite provided with norms

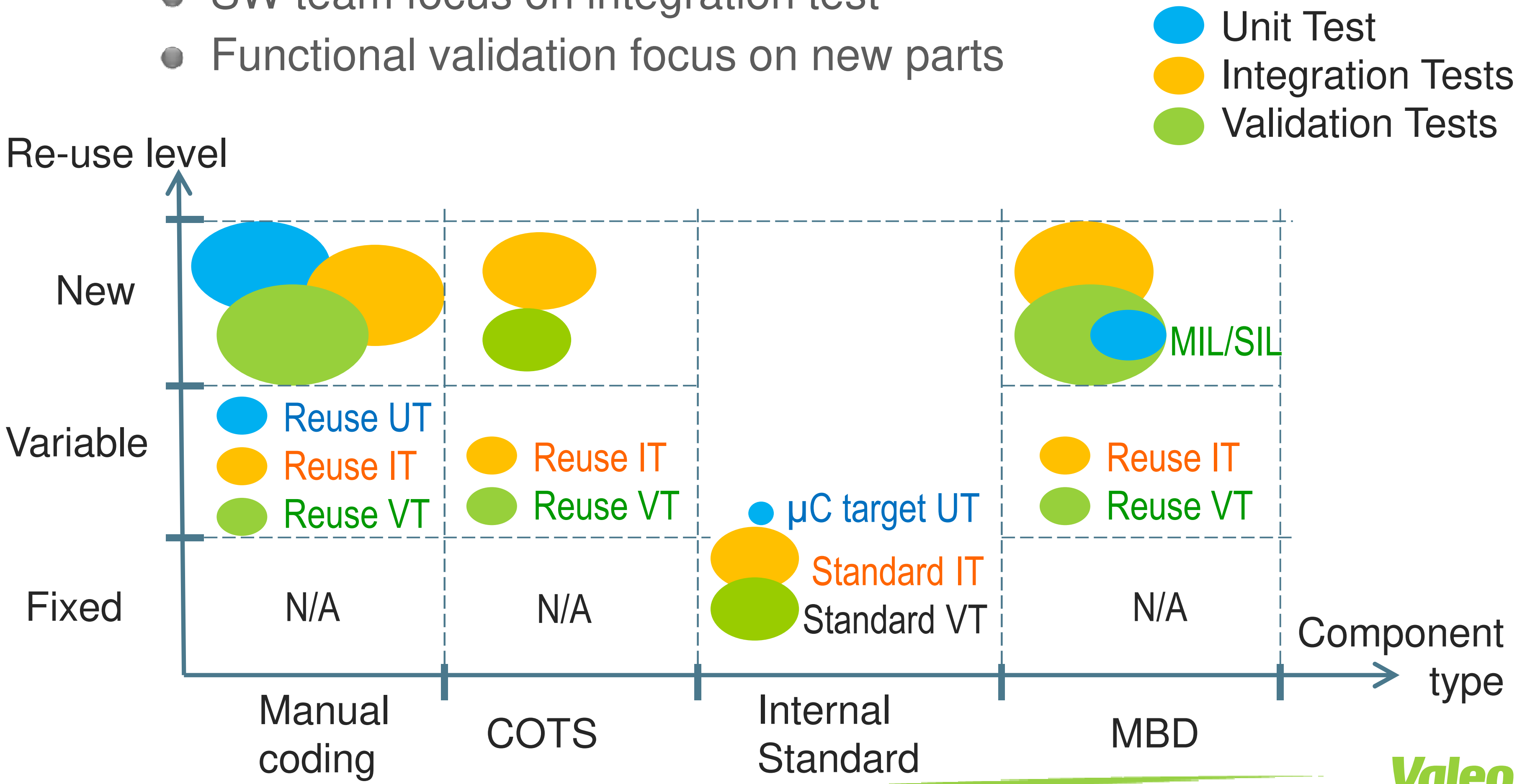
Example : Impact on Tests in MBD

- Model Based Design (MBD) component Test
  - Floating point Specification
  - Fixed point design model
    [For continuous models only]
  - Automatic code generation
    - No unit test
    - Code generator bug list check
    - Polyspace check on code generated
  - Validation played on both model
    - Functional test case design (Equivalence Class method)
    - Robustness tests
    - Structural test (MIL/SIL/PIL coverage)
  - PC Executable can be delivered by mail to Car Manufacturer for validation



Conclusion – Test effort

- Test optimization result
  - Unit Test effort divided by 10
  - SW team focus on integration test
  - Functional validation focus on new parts



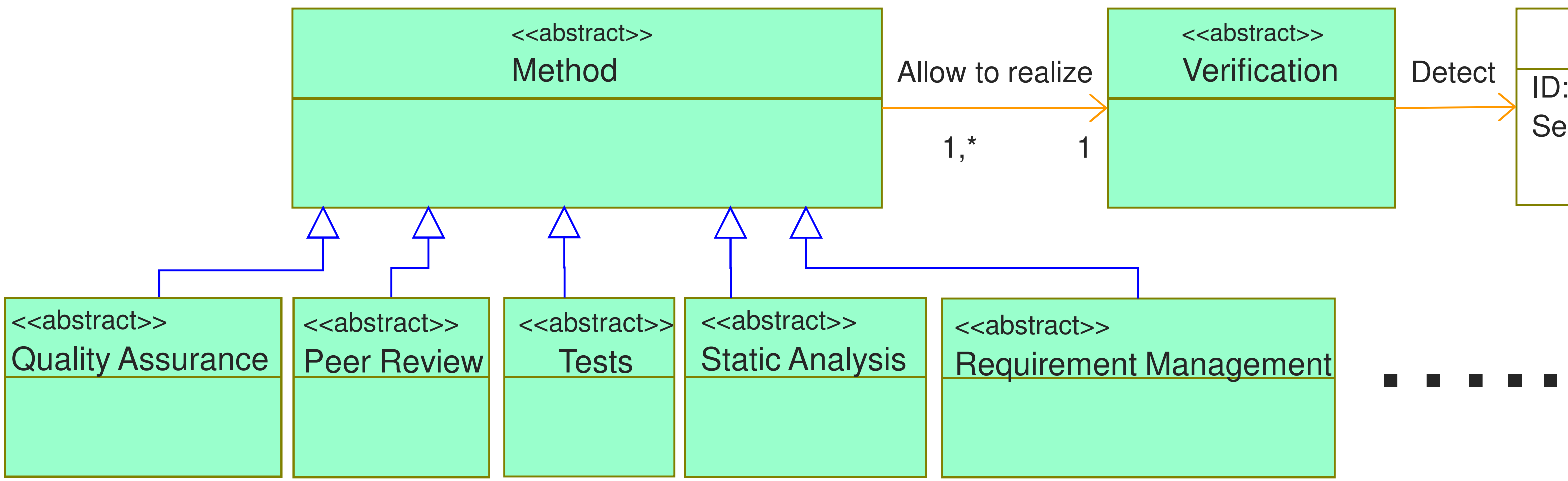
Agenda

1. Tests usual/specific issues in real-time embedded automotive software - Context
2. Optimize test effort during design process - Triptych
3. Test and verification techniques portfolio - Examples

francois-xavier.de-launet@valeo.com

Verifications Versus Validation

- Purpose
  - Validation: Demonstrate that the software product or some software component fulfills its intended use when placed in its intended environment =>“Is it the right software”
  - Verification: Ensure that the software work products meet their explicit and implicit requirements => “Is the software right”
  - To guarantee SW quality, organization focus on validation, but scope of verification methods is large…



Best verification techniques

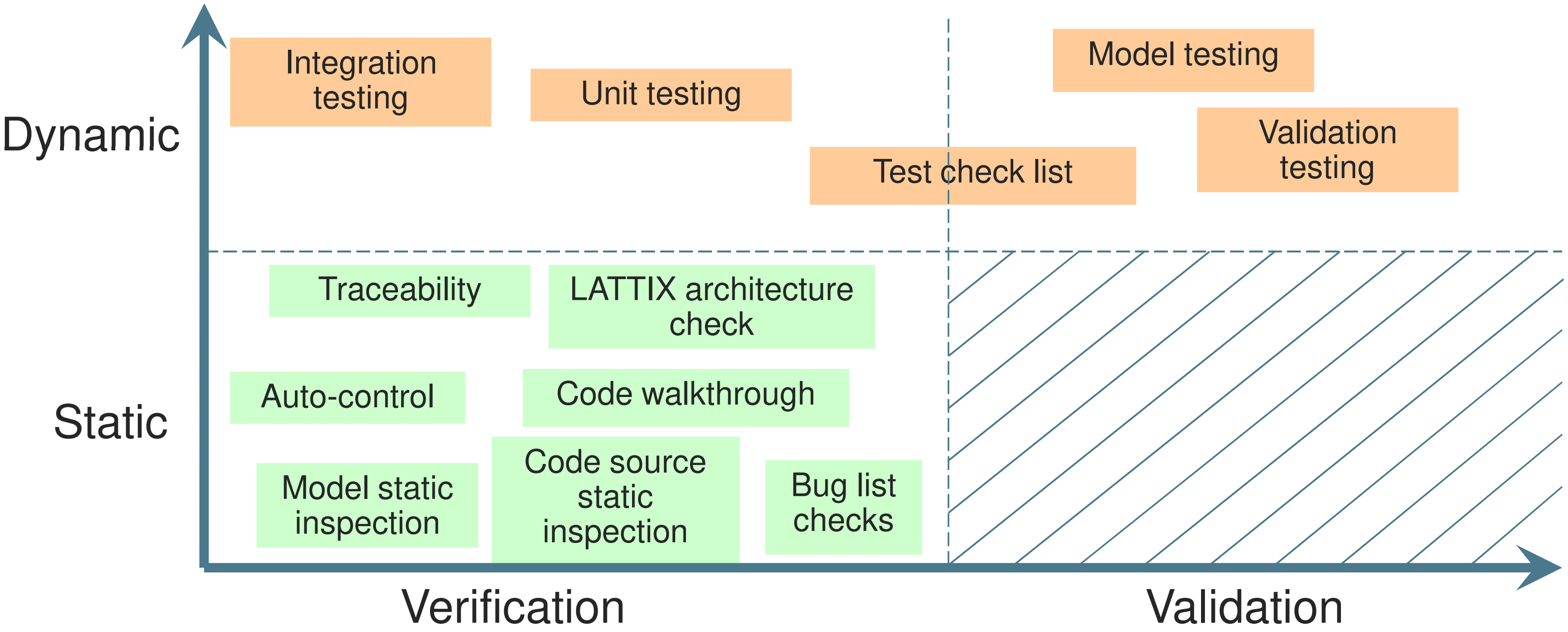
- Relative cost of fault correction *<< standard figures >>*

Stage of fault correction	**Relative cost**					
SW development plan	1					
Product requirement	1,3	1				
SW specification	2,4	1,8	1			
SW design	3,3	2,4	1,3	1		
Unit/integration test	6,8	5,1	2,8	2,1	1	
Validation tests	26	19	11	8	3,8	1
Product/Vehicle integration	96	72	39	30	14	3,7	1

- We need techniques for effective and earlier (in the lifecycle) defect removal, and technical debt limitation
  - Before coding (Peer reviews, simulation, calculation, executable spec, experience check-list)
  - Before integration (Static analysis, Unit test)

Verification techniques mapping in VALEO

- Static testing
  =>Testing of an object without execution on a computer.
- Dynamic testing
  =>Testing software through executing it.



Peer reviews - Benefits

Detect defects "at the earliest, at the cheapest"

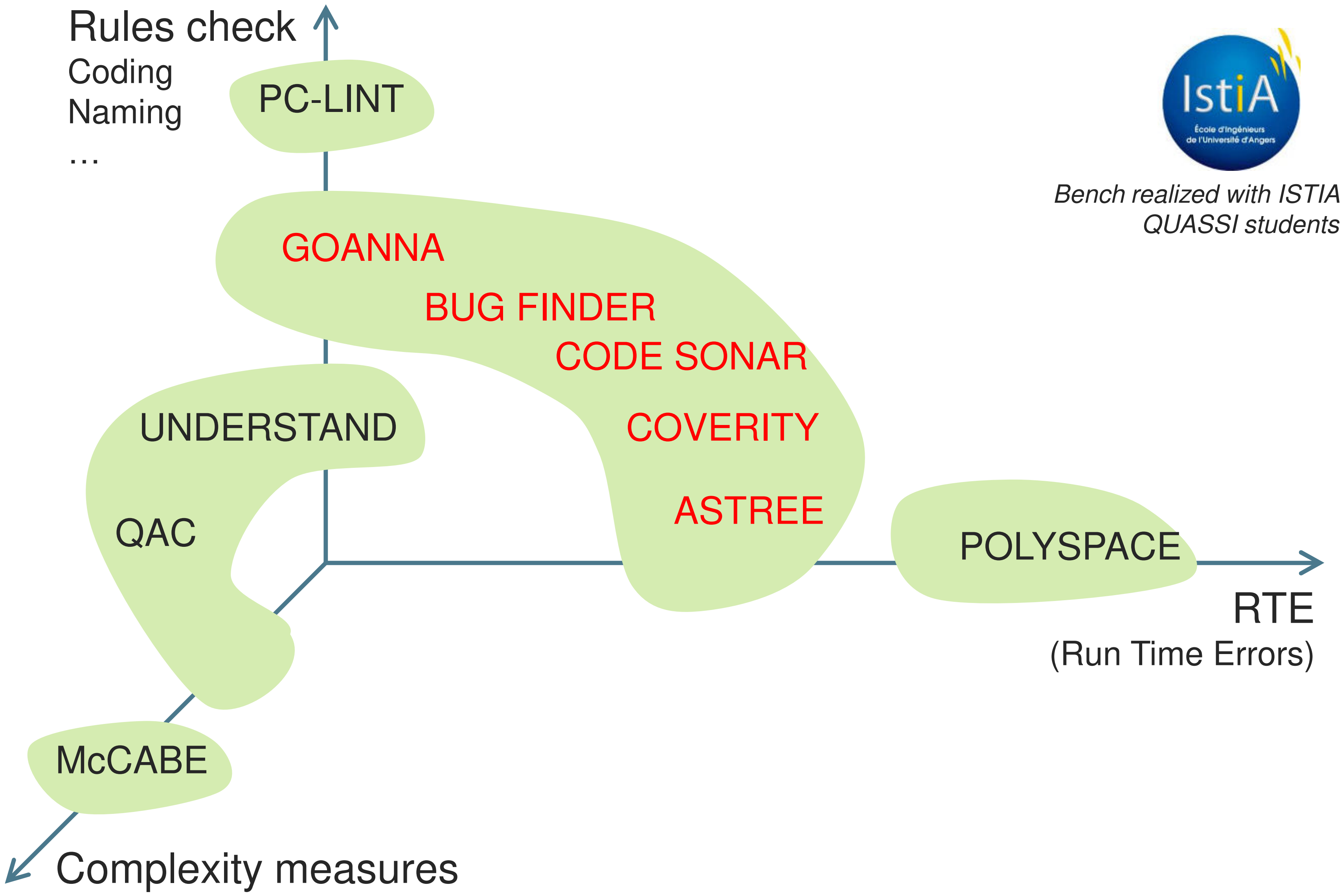
- Peer reviews should focus on design phase
  - Defect introduction: 80% in implementation, 20% in Spec/Design
- Figures
  - ISC Payback

ROI = 1 : 8,5

Read pages per hour (page/hour)	7,59
Defects Acceptance (%)	**67,5%**
Defects per page (Blocking + Major)/page	**0,25**
Defects per hour (Blocking + Major)/hour	**1,91**

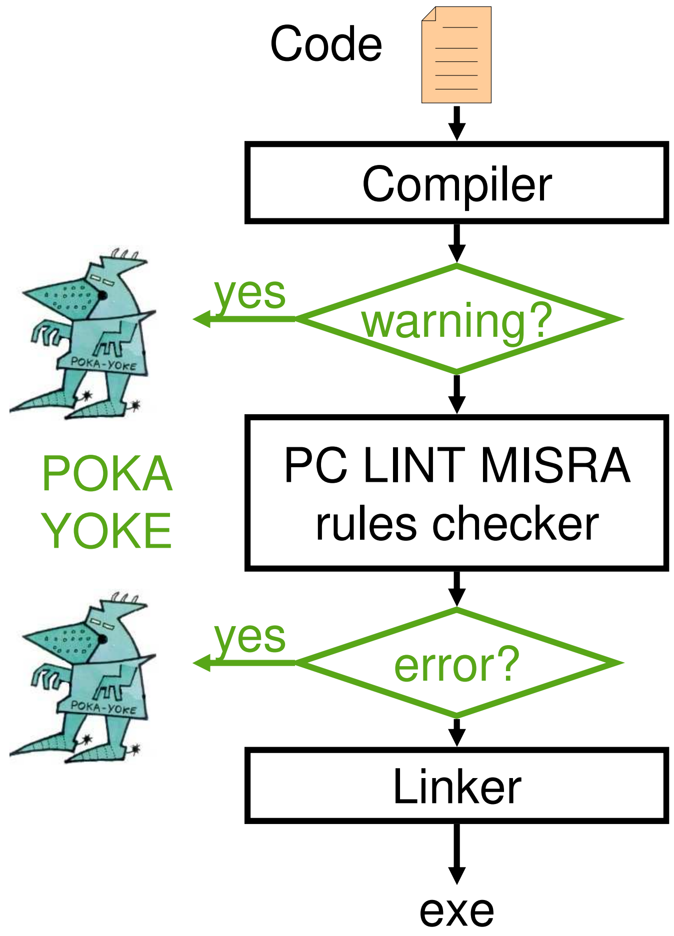
  - Hours in peer review are about 4% of SW development workforce.
- Corollary: Intangible benefits
  - Team work – Team spirit promotion - Transferring knowledge - On the Job Training
  - Culture (Embed quality assurance in process, professional attitude, open mind)
  - Standards can be improved, or come to life thru inspections

Static analysis (1/3) – 3 axes for static analysis



Static analysis (2/3) – Production chain

- Coding rules check purpose:
  - Reliability (Defensive programming)
  - Maintainability (Robust to modification)
- Use centralized IDE
  - Incorruptible
- Integrate coding rules checks in production chain
  - Max level of compiler warning
  - MISRA rules checker (PC-LINT)
  - Naming rules checker (QAC-VNR)
- Poka Yoke («*Détrompeur* »)

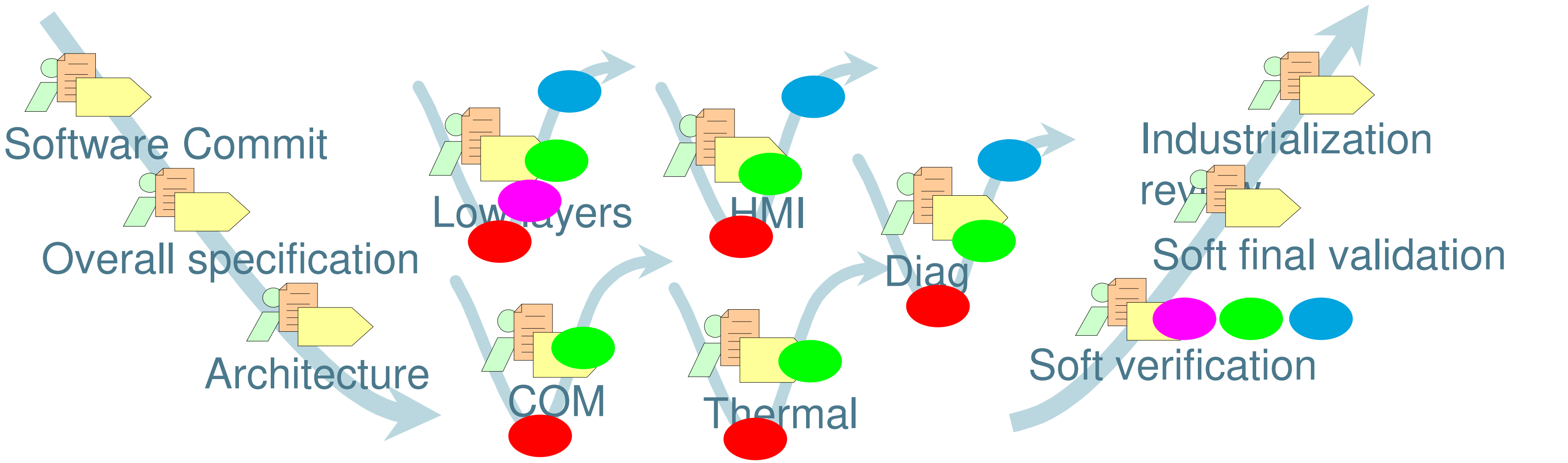


Static analysis (3/3) – Cadencing

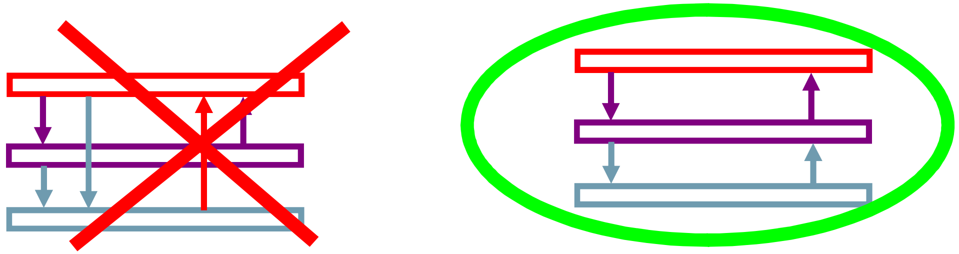
- Coding rules check ➲ IDE + Customer deliveries
- Run-Time Errors analysis ➲ Project dependant
- Assumptions made for test optimization:

  => Need to be checked along project

  - Cut complexity in architecture ➲ Architecture pattern check
  - Simple « Glue ware » ➲ Code complexity check
  - Re-use level ➲ Configuration Management check



Static analysis – Architecture pattern check (1/2)

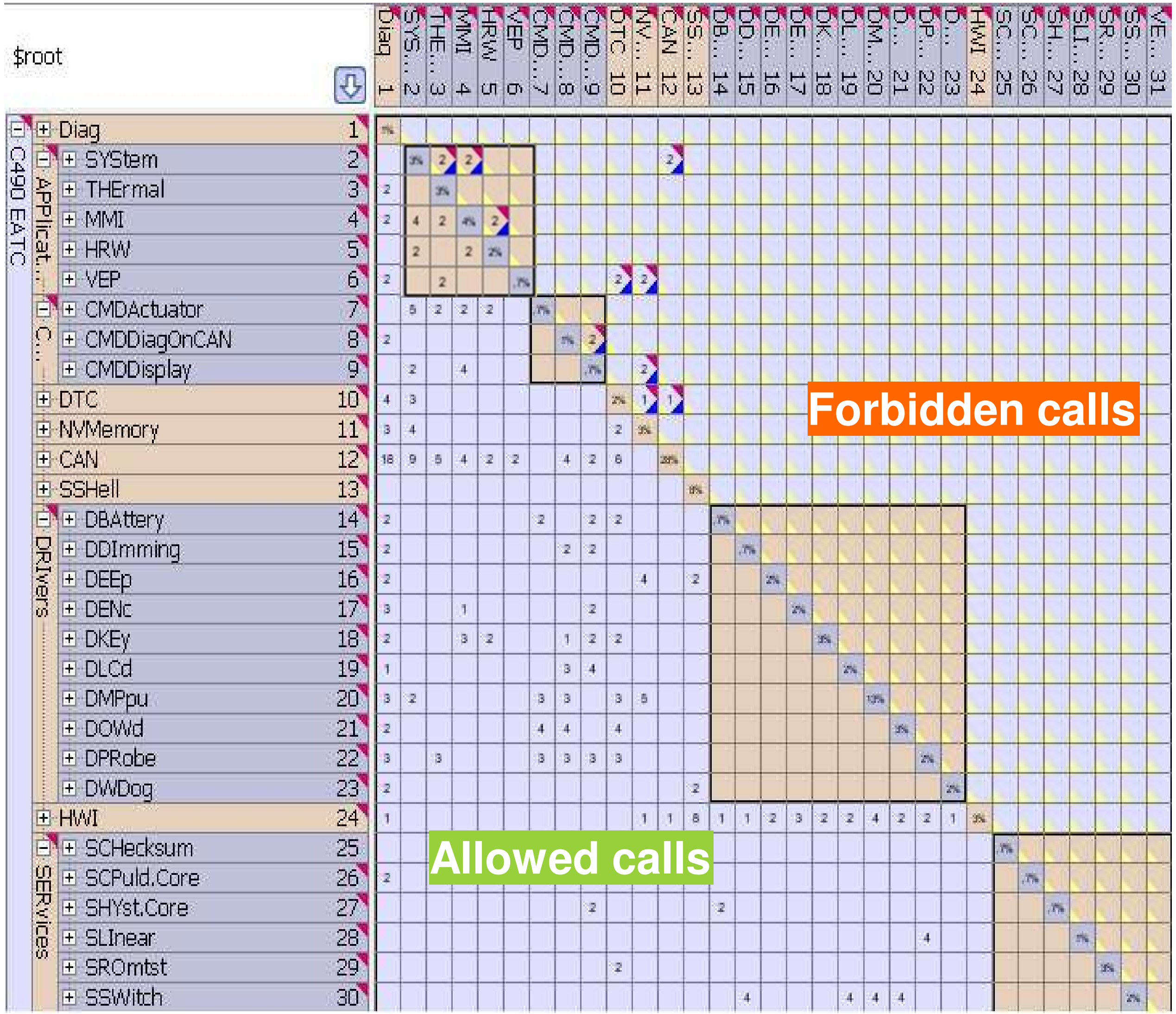


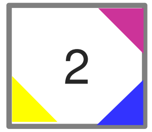
Static architecture layering

A component in the layer N provides its services to the layer N+1 and uses the services of the layer N-1

- Check done during component integration test + regression test
  - Use of "design structure matrix" (DSM) to represent the system
  - Tester can easily comprehend architecture pattern by looking at single table

Static analysis – Architecture pattern check (2/2)





Rules defined

Rules violated

Analyze done

DSM partitioning (Sub system)

LATTIX

Static analysis – Assumptions check

- Code complexity check
  - Check assumption of simple code that drives test optimization
    - Interfaces between components *(« Glue ware »)*
    - Diagnosis (Garage) functions, Hardware access µC dependant

…it can happen that interface implement treatments (Deliberate or inadvertent technical debt)

- Used criteria
  - Cyclomatic complexity + Essential complexity
  - Code destructurations
  - Code decision/condition density

McCabe SOFTWARE

- Typical outcomes (After analysis)

Metrics	Ratio	Absolute	Comment/Decision
Critical Function	7,32%	259/3539	=> Below 10% treshold
Crtical GLUEWARE function	2,13%	5/235	=> Re factor Interface
Critical MANUAL HWI function	17,65%	18/102	=> Control plan
Critical MANUAL DIAG function	4,04%	11/272	=> Control plan

Valeo

Experience check-list

« *Errare humanum est, perseverare diabolicum* »

- Capitalization of VALEO experience is based on Lessons Learned Card process:
  - Customer issues root causes are analyzed
  - Fault Tree Analysis is conduced for bug Occurrence/Non-Detection
  - Standard process is improved accordingly
- Most efficient ways:
  - Use Design Anti-pattern check list «*Betisier*»
  - Use experience test check list (« IJIWARU » tests)
    - Or « how to test what software is not supposed to do »?
  - Use all code generator bug list

Conclusion

- Optimize test strategy
  - Appropriate process (Build quality, cut project complexity)
  - Cut product complexity
  - Requirement analyze is Architecture driven
  - Promote Re-use
- Complement test with static analysis
  - Sooner the better (Before code or close to code)
  - ⚠ Do not forget to check assumption for test optimization

Automotive technology, naturally

francois-xavier.de-launet@valeo.com